Gestion de versions : démarrer avec Git et Github

Rédacteur(s) : Philippe Mariano

[Mise à jour le 10/8/2021]

- **Sources**
  - Youtube : Débuter avec Git et Github en 30 min
  - Openclassrooms : Gérez votre code avec Git et Github
  - Documentation sur le site de Git : Les bases
- **Ressources**
  - Aide mémoire Github - Git
  - “Une référence visuelle de Git” est disponible ici
- **Commandes** utilisées dans la page

config	init	add	commit	revert	status	log
branch	checkout	merge	remote	push	pull	

1. Introduction

1.1 La gestion de version

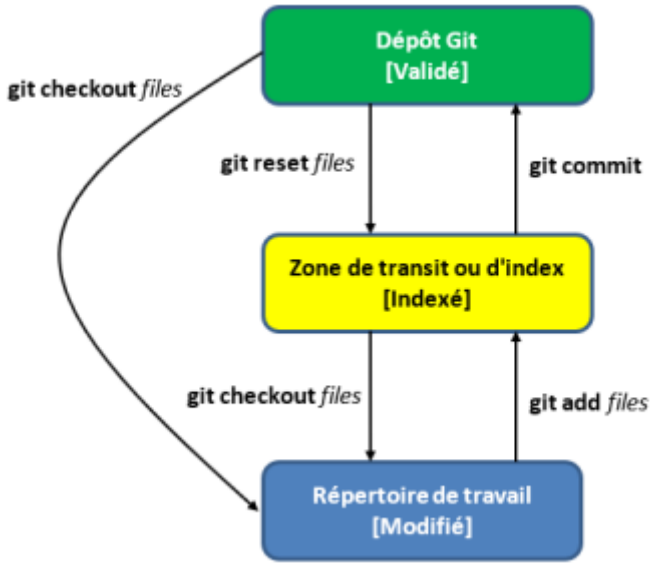
- **Présentation** de la gestion de version sur SlidePlayer et sur la documentation de Git.

1.2 Git

- **Présentation**
  Git est un système de gestion de versions. Il permet d'enregistrer et de suivre l'évolution de fichiers au cours du temps. Principalement utilisé par les développeurs, il facilite le travail collaboratif en permettant à chacun de visualiser les derniers changements, de repérer les modifications ayant pu entraîner des problèmes, etc.
  Git stocke et gère des *instantanés* de son espace de travail.
  Un de ses atouts est d'être un système de gestion décentralisé/distribué. Il ne nécessite donc

pas de de serveur central : le répertoire Git contient tout l'historique du projet et peut donc être transféré à un autre collaborateur qui disposera de tout le travail réalisé et de tout l'historique.
C'est un logiciel libre créé par Linus Torvalds, auteur du noyau Linux, et distribué selon les termes de la licence publique générale GNU version 2.
En 2016, il s'agissait du logiciel de gestion de versions le plus populaire utilisé par plus de douze millions de personnes.

- **Principe de fonctionnement**
  - Tout **projet Git** se compose de trois éléments : un **dépôt Git**, une **zone d'index** et un **répertoire de travail**. Dans un projet Git, les fichiers peuvent se trouver dans un des trois états suivants :
    - **validé** : les données sont stockées dans la base de données local.
    - **modifié** : le fichier a été modifié mais il n'est pas encore stocké.
    - **indexé** : le fichier a été marqué pour être ajouté au prochain instantané du projet.



2. Installation et paramétrage de Git

2.1 Installation

- **Sous Windows10**
  - Télécharger le logiciel et suivre les instructions sur le site de Git.
  - **Vérifier la version**
    - Entrer la commande cmd dans la barre de recherche pour ouvrir une **invite de commandes** puis tester la version avec le code ci-dessous.

```
git --version
# Exemple de résultat : "git version 2.25.1.windows.1" indique que git est correctement installé.
```

- **Sous Linux**

> Git est packagé pour la plupart des distributions Linux, on vérifiera la version avant de faire une mise à jour avec les commandes suivantes.

*.bash

```
git --version
# Exemple de résultat sur un Raspberry Pi : git version 2.20.1
sudo apt install git
# Exemple de résultat : git est déjà la version la plus récente (1:2.20.1-2+deb10u1).
```

2.2 Paramétrages

- **Paramétrage initial** à faire une seule fois après l'installation
  - **Définir son identité** avec les commandes ci-dessous pour éviter de le faire à chaque modification.

```
git config --global user.name "Prénom Nom"
git config --global user.email "mon_mail"
```

Pour connaître la configuration d'un dépôt et notamment le nom et le mail de celui qui signe les commits par défaut, entrer la commande : **git config -l** .

---

3. Git en ligne de commandes : les bases

- **Commandes utilisées** dans ce paragraphe

add	commit	revert	config	init	log	status

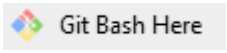
Un dépôt Git se crée dans un répertoire de travail. L'effet obtenu n'est pas visible, **.git** est un répertoire caché.

3.1 Créer un répertoire de travail

> Créer un répertoire *nom_rep* et placer un fichier *nom_fichier.ext*.

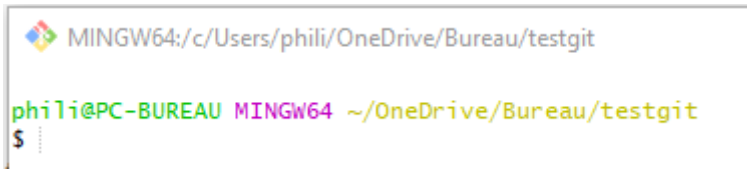
Exemple : le fichier index.html est situé dans le répertoire testgit

3.2 Se placer dans le répertoire de travail avec la console Git

- Faire un **clic-droit** dans le réperoire de travail et sélectionner 

. Une console Git s'ouvre comme dans l'exemple ci-dessous.

Exemple



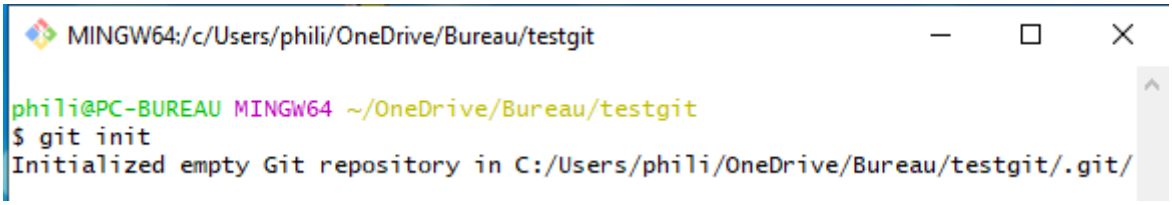
3.3 Initier un dépôt Git dans le répertoire de travail

- On utilise la commande ci-dessous pour initier un dépôt Git dans le répertoire de travail c'est à dire indiquer à Git que nous voulons faire de la gestion de version dans ce répertoire.

*.bash

```
git init
```

Exemple : dépôt Git vide !



3.4 Connaître le statut des fichiers dans Git

- On doit régulièrement vérifier le statut des fichiers dans un dépôt Git (notamment pour savoir s'ils ont été modifiés ou non depuis le dernier commit). Pour cela, on utilise la commande ci-dessous :

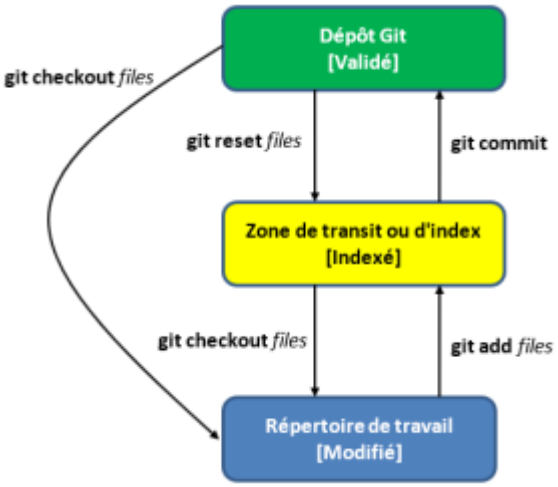
*.bash

```
git status
```

Exemple : avant d'avoir ajouté le fichier index.html à Git (les informations ci-dessous sont précisées

plus loin)

```
MINGW64:/c/Users/phili/OneDrive/Bureau/testgit

phili@PC-BUREAU MINGW64 ~/OneDrive/Bureau/testgit (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        index.html

nothing added to commit but untracked files present (use "git add" to track)
```



3.5 Faire un enregistrement du projet

- Pour faire un **enregistrement** d'un ou plusieurs fichiers dans Git, **deux étapes** sont nécessaires :
  1. **Sélectionner** le ou les fichiers à ajouter à la **zone de transit** (également appelée **zone d'index**) avec la commande **add**.
  2. **“Faire une photo”** du ou des fichiers à un instant donné avec la commande **commit** pour alimenter le **dépôt Git**.

*.bash

```
git add nomfichiers
git commit -m "texte explicatif"
```

Exemple : Après avoir écrit du code dans un fichier index.html, on l'ajoute à la **zone de transit** puis on le valide.

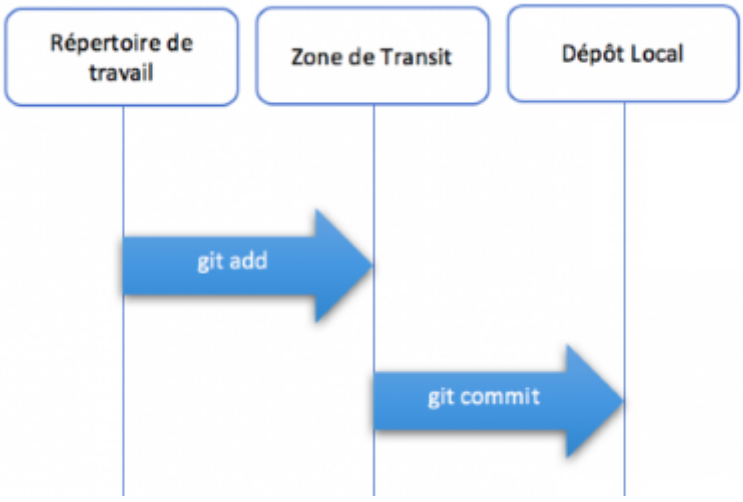
```
MINGW64:/c/Users/phili/OneDrive/Bureau/testgit

phili@PC-BUREAU MINGW64 ~/OneDrive/Bureau/testgit (master)
$ git add index.html

phili@PC-BUREAU MINGW64 ~/OneDrive/Bureau/testgit (master)
$ git commit -m "Création squelette"
[master (root-commit) 29e66ab] Cr├®ation squelette
 1 file changed, 11 insertions(+)
 create mode 100644 index.html
```

Variantes de **add** et de **commit**

*.bash

```bash
# Pour ajouter tous les fichiers du répertoire de travail dans la zone d'index
git add .
# -a demande à Git de mettre à jour les fichiers existants
# dans sa zone de transit(index)
git commit -am "texte explicatif"
# pour faire un commit sans être l'auteur du dépôt
git commit -am "texte explicatif" --author "nom<adresse_mail>"
# Pour modifier le message du dernier commit faire
git commit --amend -m "Nouveautexte explicatif"
# Pour modifier le nom de l'auteur du dernier commit
git commit --amend --author "nom<adresse_mail>"
```

Résumé



Git surveille les fichiers et non les dossiers.

3.6 Afficher l'historique des modifications

- On utilise la commande ci-dessous pour visualiser l'**historique des modifications** par ordre chronologique.

*.bash

```bash
git log # affiche tous les commits
git log -2 # limite l'affichage aux deux derniers commits
git log -p -2 # comme ci-dessus et affiche les modifications
```

Exemple : deux commits on été réalisés sur le fichier index, dans l'ordre **29e66ab** puis **12fe1e2**.

```
MINGW64:/c/Users/phili/OneDrive/Bureau/testgit

phili@PC-BUREAU MINGW64 ~/OneDrive/Bureau/testgit (master)
$ git log
commit 12fe1e2a505378f013062b94c8a42169abc48451 (HEAD -> master)
Author: Philippe Mariano <philippemariano@gmail.com>
Date:   Tue Jun 30 11:33:39 2020 +0200

    Ajout titre dans l'index

commit 29e66ab53da80469c0214d11422c7e460c48644c
Author: Philippe Mariano <philippemariano@gmail.com>
Date:   Tue Jun 30 11:16:09 2020 +0200

    Création squelette
```

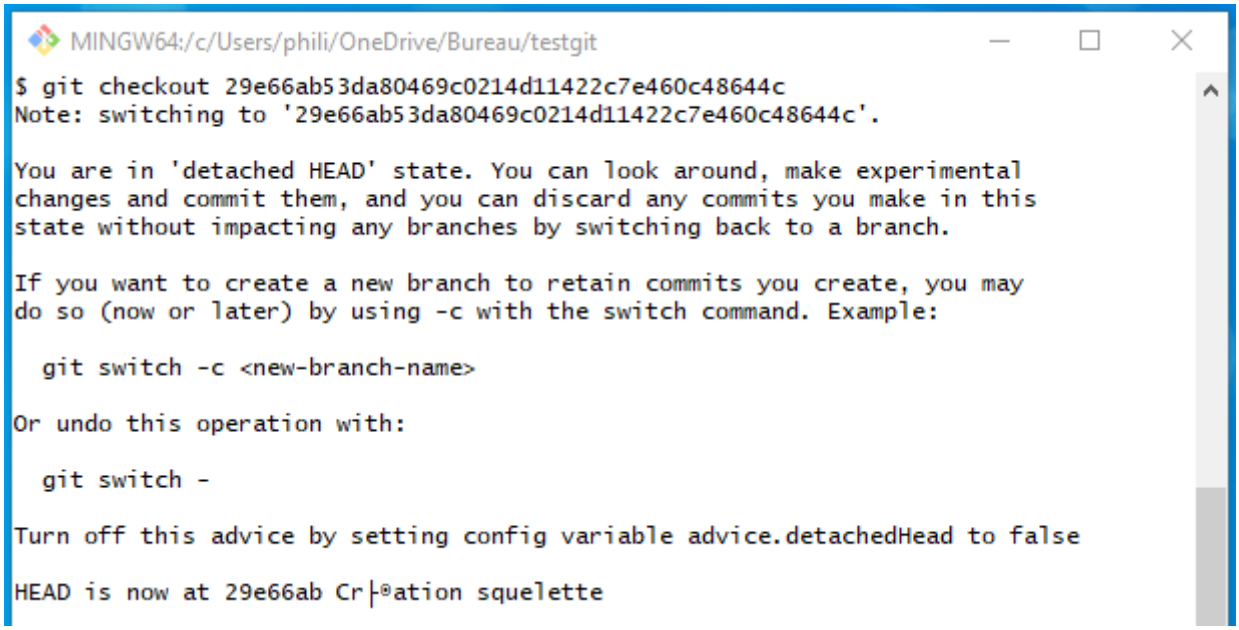
3.7 Revenir sur un commit

Il est possible de se positionner sur un commit antérieur pour par exemple trouver la source d'une erreur apparue dans le dernier commit.

*.bash

```bash
git checkout SHAduCommit #  SHA (acronyme de Secure Hash Algorithm) est associé à plusieurs fonctions de hachage
                         #  cryptographiques. Unique, il sert d'identifiant au commit.
```

Exemple : étant sur le dernier commit, on se positionne sur le premier.



Pour revenir sur le **dernier commit**, entrer : **git checkout** master (notion vue plus loin)

3.8 Annuler, modifier un commit

- **Annuler** un commit

Il est possible de changer d'avis et d'annuler un commit par :

*.bash

```
git revert SHAduCommit # annulation du commit ayant pour identifiant SHAduCommit
git revert HEAD # annulation du dernier commit
```

- **Annuler** tous les changements (possible avant d'avoir fait un commit)

*.bash

```
git reset --hard # à utiliser avec précaution
```

3.9 Travailler avec des branches

> Une branche est une copie du projet initial. Elle permet de diverger de la ligne principale (master) pour continuer à travailler sans impacter cette ligne (faire des essais, travailler à plusieurs, etc.).

- **Commandes utilisées** dans ce paragraphe

branch	checkout	merge

- **Créer** une branche

*.bash

```
git branch nombranche # Exemple : git branch backgroundcolor
```

- **Lister** les branches du dépôt

*.bash

```
git branch
```

Exemple

```
MINGW64:/c/Users/phili/OneDrive/Bureau/testgit
$ git branch backgroundcolor

phili@PC-BUREAU MINGW64 ~/OneDrive/Bureau/testgit (master)
$ git branch
  backgroundcolor
* master
```

La branche dans laquelle on se trouve est identifiée par un **asterisque**.

- **Changer** de branche

*.bash

```
git checkout nombranch # Exemple : git checkout backgroundcolor permet de passer sur la branche backgroundcolor
```

Exemple

```
MINGW64:/c/Users/phili/OneDrive/Bureau/testgit

phili@PC-BUREAU MINGW64 ~/OneDrive/Bureau/testgit (master)
$ git checkout backgroundcolor
Switched to branch 'backgroundcolor'
```

- **Fusionner** des branches
  - Pour **intégrer** les modifications faites sur une branche dérivée à la branche master :

1. **Basculer** sur la branche **master**
2. **Fusionner** master et la branche dérivée
3. **Supprimer** la branche dérivée (facultatif)

*.bash

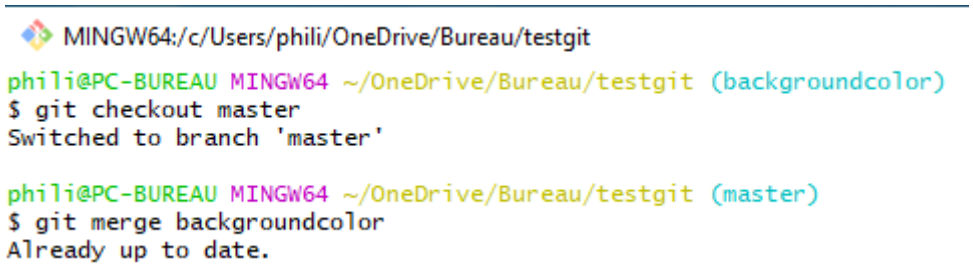
```bash
git checkout master # 1
git merge nombranche # 2
git branch -d nombranche # 3
```

Exemple : un répertoire **style**, contenant un fichier **style.css** a été ajouté sur la branche backgroundcolor du projet testgit.

1. On se place sur master et on fusionne la branche backgroundcolor avec la branche master.

*.bash

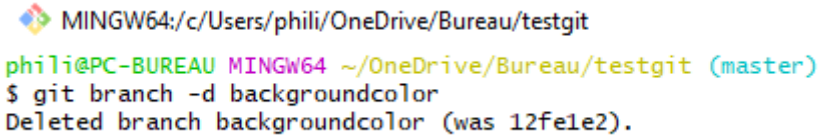
```bash
git checkout master
git merge backgroundcolor
```



2. On supprime la branche backgroundcolor

*.bash

```bash
git branch -d backgroundcolor
```



- **Résoudre** un conflit : vidéo sur OpenClassrooms.

3.10 Supprimer un dépôt git

Pour supprimer le dépôt git du répertoire de travail, il suffit de supprimer le répertoire caché **.git**. Sous W10, celui-ci est affiché en cochant *Eléments masqués* dans le menu *Affichage* de l'explorateur de fichiers.

---

4. Plus loin avec Git

4.1 Retrouver le nom de l'auteur d'une modification

- **Commandes utilisées** dans ce paragraphe

blame	**show**

> **git blame** a pour fonction générale d'afficher les métadonnées d'auteur associées à des lignes de commit spécifiques dans un fichier.

*.bash

```
git blame nom_fichier
```

Exemple : dans le fichier index.html toutes les modifications ont été faites par Philippe Mariano

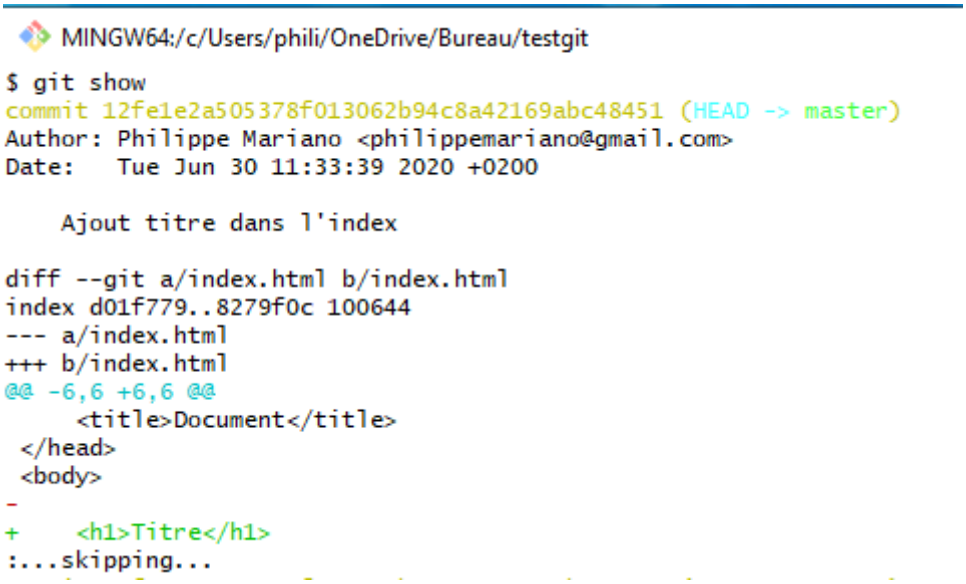
```
MINGW64:/c/Users/phili/OneDrive/Bureau/testgit
phili@PC-BUREAU MINGW64 ~/OneDrive/Bureau/testgit (master)
$ git blame index.html
^29e66ab (Philippe Mariano 2020-06-30 11:16:09 +0200  1) <!DOCTYPE html>
^29e66ab (Philippe Mariano 2020-06-30 11:16:09 +0200  2) <html lang="en">
^29e66ab (Philippe Mariano 2020-06-30 11:16:09 +0200  3) <head>
^29e66ab (Philippe Mariano 2020-06-30 11:16:09 +0200  4)     <meta charset="UTF-8">
^29e66ab (Philippe Mariano 2020-06-30 11:16:09 +0200  5)     <meta name="viewport" c
ontent="width=device-width, initial-scale=1.0">
^29e66ab (Philippe Mariano 2020-06-30 11:16:09 +0200  6)     <title>Document</title>
^29e66ab (Philippe Mariano 2020-06-30 11:16:09 +0200  7) </head>
^29e66ab (Philippe Mariano 2020-06-30 11:16:09 +0200  8) <body>
12fe1e2a (Philippe Mariano 2020-06-30 11:33:39 +0200  9)     <h1>Titre</h1>
^29e66ab (Philippe Mariano 2020-06-30 11:16:09 +0200 10) </body>
^29e66ab (Philippe Mariano 2020-06-30 11:16:09 +0200 11) </html>
```

> **git show** est un utilitaire de ligne de commande utilisé pour afficher des détails supplémentaires sur les objets Git, comme les blobs, les arborescences, les tags et les commits.

*.bash

```
git show
```

Exemple : on voit ci-dessous qu'un fichier style/style.css a été ajouté au projet et son contenu



4.2 Ignorer des fichiers

Créer un fichier **.gitignore** et y placer le nom de tous les fichiers qui ne doivent pas être indexés. Voir la vidéo sur OpenClassrooms.

4.3 Eviter des commits superflus

Voir la vidéo sur OpenClassrooms.

---

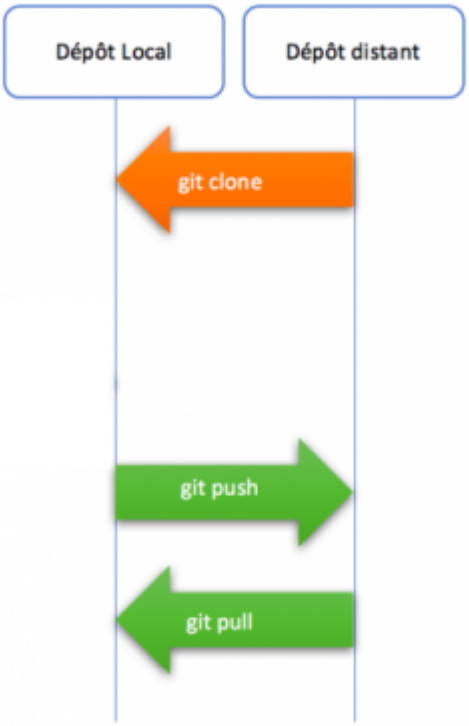
5. Collaboration sur Github

- **Source**
  - Documentation sur le site de Git : GitHub

GitHub est un service web d'hébergement et de gestion de développement de logiciels, utilisant le logiciel de gestion de versions Git.

- **Commandes utilisées** dans ce paragraphe

clone	remote	push	pull



5.1 Présentation

L'initiateur d'un projet souhaite faire participer des collaborateurs.

- **Situation de départ**
  - Les fichiers de l'**initiateur** du projet sont dans un dépôt local (sur sa machine).
  - Les futurs **collaborateurs** possèdent un **compte sur GitHub** ou peuvent en créer un et y accéder.
- L'**initiateur** du projet doit transférer ses fichiers dans un dépôt distant (sur GitHub), pour cela :
  1. Il crée un compte s'ils n'en dispose pas (**§5.2**)
  2. Il crée un dépôt vide sur Github (sans README) (**§5.3.1**)
  3. Il associe son dépôt local et le dépôt distant. (**§5.3.2**)
  4. Il transfére l'intégralité de son dépôt local dans le dépôt distant. (**§5.3.3**)
  5. Il effectue des sauvegardes, mises à jour en :
     1. Transférer ses modifications vers le dépôt distant. (**§5.5.1**)
     2. Mettant à jour son dépôt local à partir du dépôt distant. (**§5.5.2**)
  6. Il attribue les **droits d'écriture** sur le dépôt distant aux collaborateurs. (**§5.6**)
- Le **collaborateur** fait une copie du contenu du dépôt distant sur sa machine une fois le projet installé sur GitHub, pour cela :
  1. Il crée un compte s'il n'en dispose pas (**§5.2**)
  2. Il crée un répertoire sur sa machine.
  3. Il transforment son répertoire en un dépôt local avec la commande git init .
  4. Il clone le dépôt distant dans le dépôt local. (**§5.4**)
  5. Lorsqu'il dispose d'une copie du projet sur sa machine, il peut :
     1. Transférer ses modifications vers le dépôt distant. (**§5.5.1**)

2. Mettre à jour son dépôt local à partir du dépôt distant. (**§5.5.2**)

5.2 Créer un compte

- Se rendre sur https://github.com et compléter le formulaire d'inscription.

5.3 Travail préalable de l'initiateur du projet

5.3.1 Créer un dépôt distant public sur GitHub

- Sélectionner **+** dans l'en-tête du site  et compléter le formulaire pour créer le dépôt (**repository**) distant (**remote**).

Exemple

Cocher "**Initialize this repository with a README**" crée un premier fichier dans le dépôt. Cette option est à cocher **UNIQUEMENT** dans le cas où le projet **n'est pas dans un dépôt local** (sur sa machine).

Résultat

5.3.2 Associer un dépôt distant avec un dépôt local

> Il s'agit de créer un pont entre le dépôt distant (sur github) et le dépôt local (sur son poste) . Pour cela, on exécute la commande ci-dessous.

*.bash

```
git remote add origin [url_du_dépôt_créé_par_github] # Indique à Git d'ajouter le dépôt
                                                     # distant et de lui associer l'alias "origin"
                                                     # origin est une convention de Git
```

L'"url_du_dépôt_créé_par_github" se trouve en cliquant sur le bouton  sur la page du

dépôt.

Exemples

*.bash

```
git remote add origin https://github.com/WebGE/test1.git # cette commande entrée dans le répertoire test1
                                                         # relie test1 local à test1 sur GitHub
```

git remote permet de **vérifier l'association des dépôts distants** avec son projet.

*.bash

```
git remote # cette commande permet de vérifier l'association
```

5.3.3 Transférer l'intégralité de son dépôt local sur un dépôt distant

- Pour transférer les fichiers de son projet sur github, on utilise la commande **git push -u**.

Exemple

*.bash

```
git push -u origin master
```

5.4 Travail préalable du collaborateur : cloner un dépôt distant

> Il est possible de télécharger l'intégralité d'un dépôt distant sur sa machine en utilisant la commande **clone** de Git.

Pour cela :

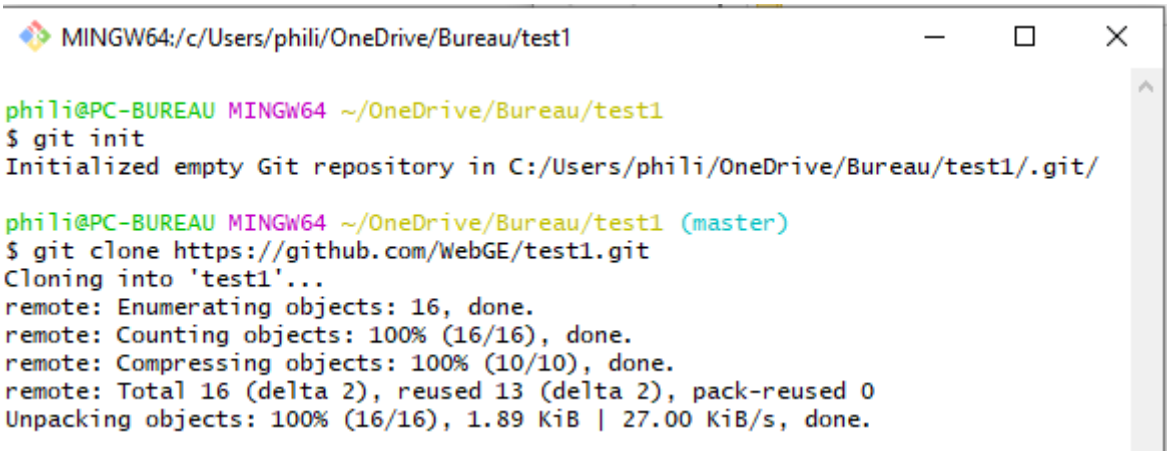
1. **Créer** un répertoire (sur sa machine) et **initier** un dépôt Git local avec git init. voir (**§3.3**)
2. Dans le dépôt local, **cloner** le dépôt distant en entrant :

*.bash

```
git clone [url_du_dépot_créé_par_github]
```

L'“**url_du_dépot_créé_par_github**” se trouve en cliquant sur le bouton 

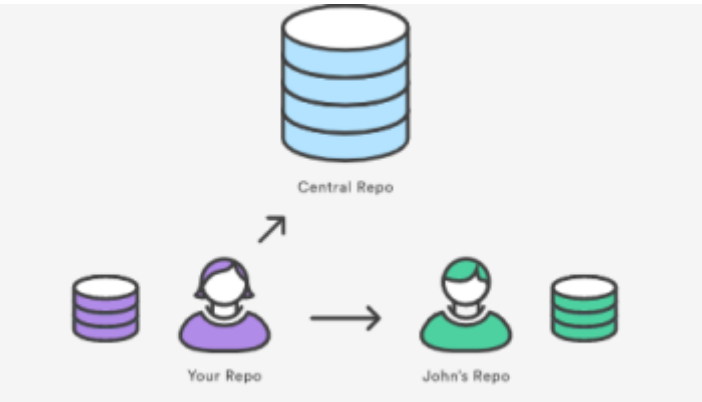
Exemple



5.5 Faire des sauvegardes, mettre à jour avec push et pull

5.5.1 Accéder à un dépôt distant

> La commande **git remote** permet de créer, d'afficher et de supprimer des connexions avec d'autres dépôts.



*.bash

```
git remote # Permet de répertorier les connexions distantes avec d'autres dépôts.
git remote -v # Idem ci-dessus mais inclut l'URL de chaque connexion.
git remote add <name> <url> # Permet de créer une connexion avec un dépôt distant.
```

```
git remote rm <name> # Permet de supprimer la connexion avec le dépôt distant nommé <nom>.
```

Exemple

*.bash

```
git remote add origin https://github.com/WebGE/test1.git # Création d'une connexion vers un dépôt sur Github
git remote add local 192.168.1.6 # Création d'une connexion avec un serveur git sur un NAS local
# Le dépôt local est connectée à deux dépôts distants
```

Exemple

```
MINGW64:/c/Users/phili/OneDrive/Bureau/tests/test1

phili@PC-BUREAU MINGW64 ~/OneDrive/Bureau/tests/test1 (master)
$ git remote
local
origin

phili@PC-BUREAU MINGW64 ~/OneDrive/Bureau/tests/test1 (master)
$ git remote -v
local   192.168.1.6 (fetch)
local   192.168.1.6 (push)
origin  https://github.com/WebGE/test1.git (fetch)
origin  https://github.com/WebGE/test1.git (push)
```

5.5.2 Mettre à jour son dépôt local à partir d'un dépôt distant.

> La commande **git fetch** télécharge des commits, des fichiers et des refs d'un dépôt distant vers le dépôt local. Le fetch est l'opération à réaliser lorsque l'on souhaite voir ce sur quoi tout le monde travaille.
>
> La commande **git pull** est utilisée pour faire un fetch du contenu d'un dépôt distant pour le télécharger, puis pour mettre à jour immédiatement le dépôt local qui correspond à ce contenu. La commande git pull est en fait la combinaison de deux autres commandes, git fetch suivie de git merge.

Exemple

*.bash

```
git pull origin master
```

A ce stade les personnes ayant téléchargé le dépôt peuvent le modifier en local, mais **ne sont pas encore autorisées** à collaborer (modifier le dépôt original sur github). Pour cela, il faut les inviter et leur accorder des droits.

5.5.3 Transférer ses modifications vers le dépôt distant

Une fois que des modifications ont été faites, ajoutées à l'index et qu'un commit à été réalisé, on peut “pousser” les modifications vers le dépôt distant si l'initiateur du projet nous a accordé les droits d'accès(voir **§5.6**).

> La commande **git push** est utilisée pour charger le contenu d'un dépôt local vers un dépôt distant.

*.bash

```
git push origin master # pour transférer le contenu du dépôt local (origin) vers la branche Master (Github)
git push origin <autrebranche> # origin <autrebranche> -> <autrebranche> distante
```

5.6 Inviter des collaborateurs et attribuer des droits

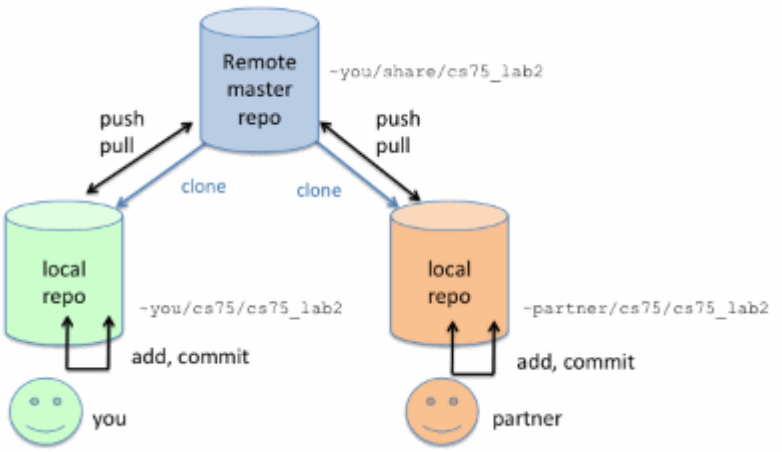
1. Sélectionner **Setting** sur la page du dépôt.
2. Sélectionner
3. Inviter un collaborateur
4. Entrer le nom du collaborateur
5. La personne sélectionnée apparaît dans la liste des collaborateurs. Elle dispose maintenant du droit d'écriture sur le dossier distant (git push)

5.7 Proposer des modifications (pull requests)

- Source : vidéo sur github

Pour proposer des modifications sur GitHub, il faut faire des **pull requests**.

1. Faire un **fork** (copie du dépôt d'un projet existant sur GitHub vers son compte Github)
2. Cloner le projet vers sa machine
3. Consulter les conditions pour réaliser une pull request dans le readme du projet
4. Faire une modification et la pousser sur GitHub
5. Faire une pull request



6. En résumé

- Pour faire de la gestion de versions **sur sa machine**
  1. Télécharger et installer Git
  2. Initialiser Git dans son répertoire de travail avec un (**git init**)
  3. Suivre le cycle : modification(s) → ajout des fichiers à la zone d'index (**git add**) → validation (**git commit**)
  4. Créer des branches (**git branch** *nombranche*) et fusionner (**git merge** *nombranche*)
- Pour **collaborer à un dépôt**
  1. Créer un compte sur GitHub
  2. Cloner un dépôt (**git clone** *url_du_dépot_distant*)
  3. Rapatrier sur sa machine les modifications du dépôt distant (**git pull** *origine nom_branche*)
  4. Publier ses modifications (**git push** *origine nom_branche*) sur le dépôt distant.